
ICARO
Nuestro compromiso es con el *futuro*.

Back End

Clase 7

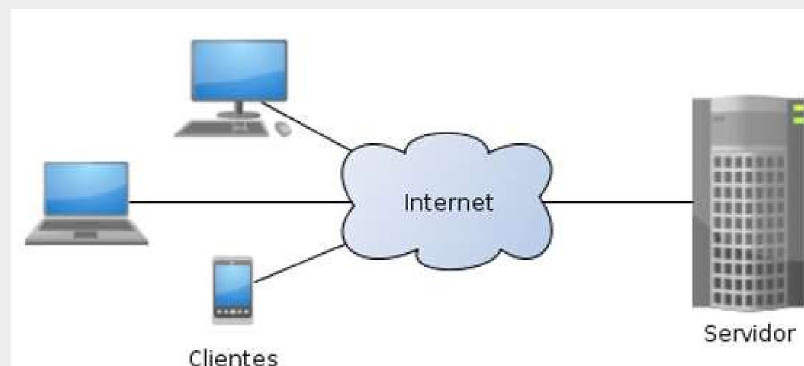
¿Qué veremos hoy?

Hoy vamos a continuar con el módulo de **Back-End**, donde veremos como usar **sesiones** y **cookies** con **Express**. También analizaremos cómo hacer uso del módulo bcrypt para generar un **hash**. Finalmente aplicaremos estos conceptos mediante el desarrollo de una app con login.

Sesiones y Cookies

La necesidad de hacer uso de **sesiones** y **cookies** viene a raíz de que el protocolo HTTP, no mantiene estado. Es decir, a nivel HTTP, el protocolo no realiza un procesamiento persistente de las solicitudes y respuestas enviadas. Esto significa que debe iniciar sesión nuevamente cada vez que actualice, lo cual, a nivel usuario, nos genera una mala experiencia.

Una de la principales diferencias entre una sesión y cookie, es el lugar donde se almacena. Una sesión se almacena del lado del servidor mientras que una cookie, de lado del cliente.



Sesiones y Cookies. Funcionamiento

Cómo funciona una sesión? Cuando el cliente por ejemplo, realiza una petición para ingresar a un sistema (ejecuta un login), el servidor va a crear una **sesión** y a guardar la misma en el servidor. Cuando este último responde al cliente (por ejemplo, un navegador), este envia una **cookie**. Esta contendrá un **id** único que es el mismo que se guardó antes, de lado del servidor.

Esta cookie, sera enviada en cada peticion al servidor. Que se logra con esto último?
Mencionamos antes que HTTP, es un protocolo sin estado, mediante el mecanismo mencionado ahora, podremos tener una sesión establecida.

Sesiones y Cookies con Express

Como hemos venido utilizando y creando nuestro servidor, no se guardarán ni trataran cookies o sesiones, a menos que se indique lo contrario; es decir, de momento, al momento de crear un servidor con express, haciamos:

```
index.js > ...
const express = require('express');
const app = express();
const PUERTO = 3000;

app.listen(PUERTO, (req, res) => {
    console.log("Servidor corriendo en puerto ", PUERTO)
})
```

Para poder cambiar el comportamiento de nuestro servidor, podemos hacer uso de modulos extras, los cuales se instalan con nuestro gestor de dependencias npm.

Sesiones y Cookies con Express

Un modulo, que podemos instalar, para el manejo de sesiones en express, es express-session
https://www.npmjs.com/package/express-session

La cual se instala ingresado al terminal: **npm install express-session**

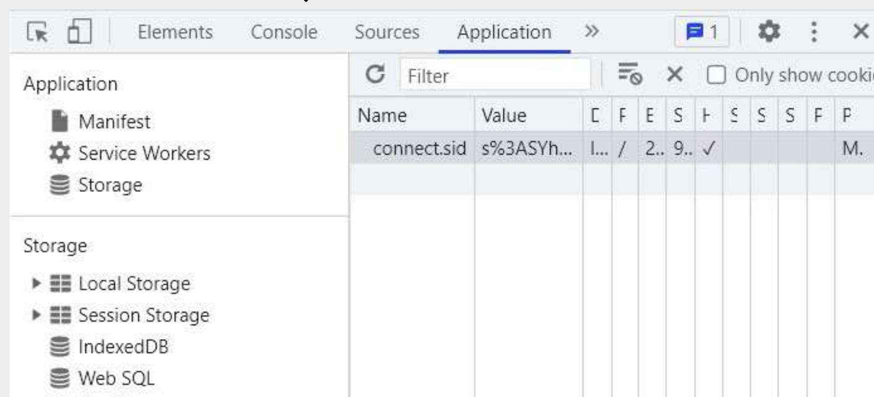
En lo que respecta al módulo a utilizar para la generación de la cookie, usaremos cookie-parser, y se instala ingresando en el terminal **npm install cookie-parser**
https://www.npmjs.com/package/cookie-parser

Poniendo en práctica lo visto

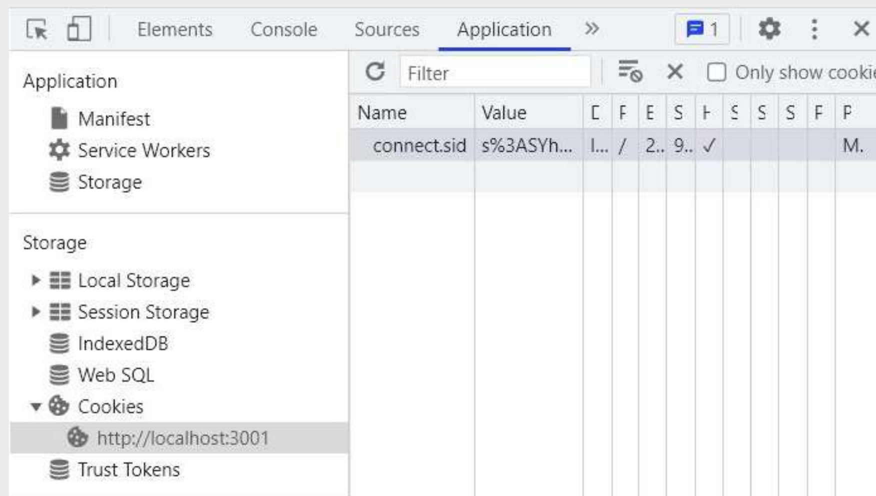
Ya hemos introducido el concepto de session y cookie, un tipico caso de uso, es el manejo de un formulario de login, donde introducimos nombre y password, al menos en este caso lo mantendremos simple.

Dentro de nuestra app, tendremos un usuario ya valido, a fin de validar lo que ingreso el usuario.
Tenemos en este caso dos posibles escenarios:

1) Se ingresa un usuario valido. En este caso, dentro de nuestro cliente (nuestro navegador), una sesion valida.



Poniendo en práctica lo visto



El nombre es provisto por express, pero vemos una cookie generada la cual se intercambiara con el servidor en cada sesión, evitando asi que tenga que ingresar sus credenciales cada vez.

Poniendo en práctica lo visto

2) El otro escenario es que se introduzca un usuario invalido. En este caso redirigimos a una vista error como ya lo hemos venido implementando.

Cabecera de nuestra aplicacion

Usuario invalido ......

volver a formulario

© Copyright 2022 Curso Full Stack Icaro

Importando nuestros módulos

Los módulos express-sesion y cookie-parser se usan de la siguiente manera:

```
const cookieParser = require("cookie-parser");
const sessions = require('express-session');
const unDia = 1000 * 60 * 60 * 24;
```

Luego, se definen dentro de express, mediante el uso de **middlewares**, esto es, un función común para todas las ejecuciones, y se realiza de la siguiente forma:

```
app.use(sessions({
  secret: "123456",
  saveUninitialized:true,
  cookie: { maxAge: unDia },
  resave: false
}));
app.use(cookieParser());
```

Detalles....

Vemos que nuestro módulo express-session requiere de varias propiedades, las cuales pueden ser consultada en https://www.npmjs.com/package/express-session para aun tener mas detalle, pero podemos mencionar de momento:

Secret: es un string unico. De momento estará en nuestra aplicación y puede ser cualquier conjunto de caracteres, aunque se debería de almacenar de forma segura en un ambiente de producción.

Resave: Toma un valor de tipo boolean (true o false). Permite que la sesión sea almacenada.

saveUninitialized: Esto permite que una sesión no iniciada sea almacenado.

Cookie: Se usa para setear el periodo de expiración de la cookie, la que es usada de lado cliente.

Esta última para este ejercicio, ha sido creada para que tenga un valor de 1 dia.

Usando hashes……

Como ya hemos mencionado, la idea de la práctica a realizar, es la de tener un formulario donde se ingresa nombre/usuario y password. Por cuestiones de seguridad, el password no debería de manejarse en texto plano, por lo que se introduce el concepto de hash.

Un hash es un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija.

Para el ambiente de express y nodejs, tenemos un módulo que nos ayuda a aplicar ya este algoritmo, y es el módulo **bcrypt**, el cual se instala ingresando al terminal npm install **bcrypt**.

Usando hashes……

Dentro de nuestra aplicación:

```
const bcrypt = require("bcrypt");
```

Listo!, ya importamos nuestro módulo, y esta listo para ayudarnos a encriptar nuestro password.

Por cuestiones de simplicidad, aun no existe base de datos, las cuales sera agregado en futuras clases; de momento tenemos solo arrays y objetos ya predefinidos.

Para el caso de esta aplicación, tenemos un usuario ya definido con los siguientes datos:

Usando hashes……

```
let USUARIO_SESION_VALIDO = {
  nombre: "usuario",
  password: "$2b$05$TqGRSuMWPledy2QgMBnG8ujh/CWypwZL/29VDNqkarkq6pbM8Q526"
}
```

Luego, tendremos el valor del password que ingresa el usuario desde el navegador. Primero, generamos el hash del password que se ingresa y se compara con el hash ya presente en por ejemplo una base de datos. Si ambos valores coinciden, el usuario es válido.

Bcrypt, nos ayuda a validar esto con una función:

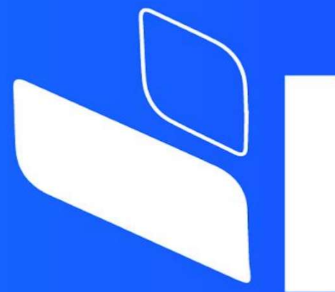
```
  var esValidoElPasswordHasheado = bcrypt.compareSync(password, passwordEncriptadoEsperado);
```

Siendo el segundo parámetro el password ya obtenido de la base de datos. El resultado de dicha comparación es un valor de tipo boolean. True si es valido o false caso contrario.

En caso de existir dudas, concurramos a clases de consulta!!

¡Vamos al código!

Muchas gracias!



ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar